Cut-and-Choose Bilateral Oblivious Transfer and Its Application in Secure Two-party Computation

Han Jiang, Xiaochao Wei, Chuan Zhao, and Qiuliang Xu

School of computer science and technology, Shandong University
jianghan@sdu.edu.cn
weixiaochao2008@163.com
zhaochuan.sdu@gmail.com
xql@sdu.edu.cn

Abstract. In secure two-party computation protocols, the cut-and-choose paradigm is used to prevent the malicious party who constructs the garbled circuits from cheating. In previous realization of the cut-and-choose technique on the garbled circuits, the delivery of the random keys is divided into multiple stages. Thus, the round complexity is high and the consistency of cut-and-choose challenge should be proved.

In this paper, we introduce a new primitive called cut-and-choose bilateral oblivious transfer, which transfers all necessary keys of garbled circuits in one process. Specifically, in our oblivious transfer protocol, the sender inputs two pairs (x_0, x_1), (y_0, y_1) and a bit τ; the receiver inputs two bits σ and j. After the protocol execution, the receiver obtains x_τ, y_σ for $j = 1$, and x_0, x_1, y_0, y_1 for $j = 0$.

By the introduction of this new primitive, the round complexity of secure two-party computation protocol can be decreased; the cut-and-choose challenge j is no need to be opened anymore, therefore the consistency proof of j is omitted. In addition, the primitive is of independent interest and could be useful in many cut-and-choose scenarios.

Keywords: Secure Two-party Computation, Round Complexity, Cut-and-choose Inverse OT, Cut-and-choose Bilateral OT

1 Introduction

1.1 Background

Secure two-party computation considers the setting where two distinct parties P_1 and P_2 holding secret inputs x and y want to compute a desired function $f(x, y)$ such that each party obtains an output, and it satisfies some security requirements. These requirements contain privacy, correctness, independence of inputs, guaranteed output delivery and fairness (see [1]). The security defined for two-party computation is based on **ideal/real simulation paradigm** [1–3], where in the ideal world there exists a trusted third party who receives inputs of the participants and sends corresponding outputs to each party, but in the real world the participants run some actual protocols. A protocol is said to be secure if no adversary can do more harm in a real execution than in an execution that takes place in the ideal world. For a specific protocol, we judge its security for a function by comparing the outputs of the adversary and honest party in a real execution to their outputs in the ideal world. For the reason of the different adversarial behavior, we mainly consider the following three kinds of model: semi-honest model, malicious model and covert model. For more details see the chapter 3,4,5 in [1].

1.2 The Cut-and-choose Technique for Secure Computation

The primary work of secure two-party computation proposed by Yao [16] is secure only in semi-honest model, the cut-and-choose technique used for constructing secure protocols in the presence of malicious adversary was firstly proposed by Lindell and Pinkas [4]. The basic idea behind cut-and-choose is that P_1 constructs many copies of the garbled circuit computing the desired function, then P_2 instructs P_1 to open a subset of these circuits (called check-circuits) and checks them, if any misbehavior is detected in check-circuits, P_2 aborts the protocol. The remaining circuits (called evaluation-circuits) are evaluated to derive the final output.

The results of all of the evaluation-circuits may be different because the faked circuits can be selected as evaluation-circuits. To resist a so called selective failure attack [9], an efficient strategy is that P_2 takes the majority outputs of evaluation-circuits as result, then P_1 can only cheat if the majority of the unopened circuits are incorrect, while all of the opened circuits are correct. The majority outputs are correct except with negligible probability, which is related to s (the number of circuits P_1 constructs), i.e, in $[4, 8, 5, 7, 6]$ the error probability respectively are $2^{-s/17}$, $2^{-s/4}$, $2^{-0.311s}$, $2^{-0.32s}$ and 2^{-s}. In [8] it shows that the bottleneck factors influencing the efficiency are the number of circuits and commitments used for consistency check.

In [5] a new technique called cut-and-choose OT was given and the number of the check circuits is just half of all circuits, whose error probability is $2^{-0.311s}$; in [7] the number of check-circuits is 60% of all circuits, which leads the probability to be $2^{-0.32s}$.

In Crypto 2013, Lindell presented a protocol in [6] satisfying that the error probability is 2^{-s}. Comparing with the protocol in [5], the new protocol's advantage reflects in the following aspects. In his protocol, P_1 and P_2 run a batch single-choice cut-and-choose OT to transfer the garbled circuits, and then P_2 evaluates the evaluation-circuits at first which is opposite to [5]. Through this process, if P_1 has cheated, P_2 can obtain two different values in one output wire of a evaluation-circuit (we call it a proof), then P_2 participates another small protocol using this proof aiming at obtaining P_1's input for computation. After the above steps, P_2 checks the check-circuits and aborts the protocol if there exists any mistakes. This changes the awkward situation in previous work that though P_2 knows that P_1 is cheating, but he can do nothing about it. This protocol gives the best error probability 2^{-s} until now, but in the small protocol there needs additional two-party computation circuits to detect the cheating (verify the correctness of P_2's proof).

At the same conference, Yan Huang et al. introduce the idea of symmetric cut-and-choose protocol [11], in which each party generates κ circuits to be checked by the other party, and then evaluates the remaining garbled circuits of the other party. The final outputs are combined by both parties' results. Compare with the one-side cut-and-choose two-party computation, where P_1 can only cheat if the majority of the evaluation circuits are incorrect and all of the check circuits are correct, a malicious party in Yan's protocol can successfully cheat only if they generate exactly $\kappa - c$ fake garbled circuits and none of them is checked by the other party. When setting $c = \kappa/2$ (which minimizes the cheating probability), the error probability reaches $2^{-\kappa+o(\log\kappa)}$.

1.3 Motivation and Contributions

In this paper, we introduce a new primitive called cut-and-choose bilateral oblivious transfer, which is inspired by the work of Lindell and Pinkas [5], in which a primitive called cut-and-choose oblivious transfer was presented and used in secure-two-party computation to intertwine the oblivious transfer and the circuit checks, and solved the selective failure attack problem. The motivation of our primitive, however, is to further improve the efficiency of secure two-party computation protocol in respect of round complexity. Specifically, our work can reduce the round complexity of the cut-and-choose phase in the protocol to only one round.

In previous works mentioned above, the delivery of the keys associated with the input wires in garbled circuits is divided into multiple stages. Thus, the round complexity is high and the consistency of challenge set $\mathcal{J}$ should be proved. In this paper, we construct an oblivious transfer protocol that can transfer all necessary keys of garbled circuits in one process. Specifically, in our oblivious transfer primitive, the sender inputs two pairs (x_0, x_1), (y_0, y_1) and a bit τ. The receiver inputs two bits σ and j. Then, the receiver obtains x_τ, y_σ for $j = 1$, and x_0, x_1, y_0, y_1 for $j = 0$. In traditional OT_1^2 protocol, even the cut-and-choose OT protocol mentioned above, the sender is passive in the sense that he prepares the garbled keys for receiver to choose. Our new primitive works in a different pattern in which the sender is active in the sense that he also decides which part of the keys should be obtained by the receiver.

By the introduction of this new primitive, we obtain a number of benefits:

- The round complexity of secure two-party computation protocol is decreased;
- The challenge set $\mathcal{J}$ is no need to be opened anymore, therefore the consistency proof of $\mathcal{J}$ is omitted;
- This primitive is of independent interest and could be useful in many cut-and-choose scenarios, not just in secure two-party computation.

In this paper, we give the efficient construction of cut-and-choose bilateral oblivious transfer based on decisional Diffied-Hellman assumption. The construction is divided into two stages. At first, we "inverse" the right of key choice from R to S in cut-and-choose OT; then we combine the cut-and-choose OT with cut-and-choose inverse OT to form cut-and-choose bilateral OT. Our cut-and-choose bilateral oblivious transfer protocol is secure against malicious adversaries and the security is proven under the standard ideal/real simulation paradigm.

1.4 Organization

We present some preliminaries such as garbled circuits and $RAND$ function in Section 2 and extract a simplified version of cut-and-choose OT in Section 3. Then we give a detailed construction and security proof of cut-and-choose inverse OT protocol in Section 4 and cut-and-choose bilateral OT protocol in Section 5. At last, we show the application of cut-and-choose bilateral OT in secure two-party computation in Section 6.

2 Preliminaries and Notations

2.1 Notations of Cut-and-choose Two-party Computation

Functionality and Inputs. Let f be a polynomial-time functionality, and let x is P_1's input and y is P_2's input. For simplicity, we assume that the input length, output length and the security parameter are all of the same length n, then we write the binary form of the inputs as $x = \tau_1\tau_2\ldots\tau_n$ and $y = \sigma_1\sigma_2\ldots\sigma_n$.

Garbled Circuits. Let $C(x, y)$ is a boolean circuit that computes the function f, that receives two inputs $x, y \in \{0,1\}^n$ and outputs $C(x, y) \in \{0,1\}^n$. The circuit $C(x, y)$ is computed gate by gate. Each gate can be represented by a function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$, so it has two input wires and one output wire. Let m be the number of all wires in the circuit C, and let $w_1, \ldots, w_m$ be the labels of these wires. In these m wires, let $w_1, \ldots, w_n$ be the circuit-input wires corresponding to input x, and $w_{n+1}, \ldots, w_{2n}$ be the circuit-input wires corresponding to input y. For each wire w_i $(i = 1, \ldots, m)$, randomly choose two independent keys k_0^i, k_1^i. Given these keys, we can compute four garbled values of each gate g whose input wires are w_i and w_j, and output wire is w_l as:

$$c_{0,0} = E_{k_0^i}(E_{k_0^j}(k_{g(0,0)}^l)), c_{0,1} = E_{k_0^i}(E_{k_1^j}(k_{g(0,1)}^l)),$$

$$c_{1,0} = E_{k_1^i}(E_{k_0^j}(k_{g(1,0)}^l)), c_{1,1} = E_{k_1^i}(E_{k_1^j}(k_{g(1,1)}^l)).$$

where E is from a private-key encryption scheme (G, E, D). The results of random permutation of above values are called garbled table for gate g. For every circuit-output wire w_t, the table consisted of the values $(0, k_0^t)$ and $(1, k_1^t)$ is called the output table (or decryption table). Then all the garbled tables and the output tables form the entire garbled circuit of C, denoted by $G(C)$.

In malicious adversaries model, for the same circuit of C, we should independently generate s garbled circuits, denoted by $G^1(C), G^2(C), \ldots, G^s(C)$. We denote the keys associated with ith wire of $G^j(C)$ as $k_0^{i,j}, k_1^{i,j}$.

The Partition of Garbled Circuits. For s garbled circuits, we use a s-bits binary string $j = j_1 j_2 \ldots j_s$ to divide them. For all $i = 1, 2, \ldots, s$, if $j_i = 0$, then the circuit $G^i(C)$ is a check-circuit, otherwise, it is a evaluation-circuit.

2.2 Randomization Algorithm *RAND* and Its Properties

In [12], Peikert et al. proposed a randomization algorithm that is based on DDH assumption as follows.

Algorithm 1: $RAND(g, h, \tilde{g}, \tilde{h})$

Let $(\mathbb{G}, g_0, q)$ be such that $\mathbb{G}$ is a group of prime order q, with generator g_0, and $g, h, \tilde{g}, \tilde{h} \in \mathbb{G}$.
Choose $s, t \leftarrow \mathbb{Z}_q$ independently, compute $u = g^s \cdot h^t$ and $v = (\tilde{g})^s \cdot (\tilde{h})^t$.
Output (u, v).

Algorithm $RAND$ has some properties and Lindell described them in [6] as follows.

Proposition 1. *Let $(\mathbb{G}, g_0, q)$ be as above and the input to $RAND$ is a DH-tuple. Let $g, h \in \mathbb{G}$ and $a \in \mathbb{Z}_q$. Then, for $(u, v) \leftarrow RAND(g, h, g^a, h^a)$, it holds that $u^a = v$.*

Proof. By the definition of algorithm $RAND$, $u = g^s \cdot h^t$, $v = (g^a)^s \cdot (h^a)^t = g^{a \cdot s} \cdot h^{a \cdot t}$. Then

$$u^a = (g^s \cdot h^t)^a = g^{a \cdot s} \cdot h^{a \cdot t} = v.$$

Proposition 2. *Let $(\mathbb{G}, g_0, q)$ be as above and let $g, h, \tilde{g}, \tilde{h} \in \mathbb{G}$. If $(g, h, \tilde{g}, \tilde{h})$ is not a DH-tuple, then the distributions $(g, h, \tilde{g}, \tilde{h}, RAND(g, h, \tilde{g}, \tilde{h}))$ and $(g, h, \tilde{g}, \tilde{h}, g_0^\alpha, g_0^\beta)$ are equivalent, where $\alpha, \beta \leftarrow \mathbb{Z}_q$ are random.*

Proof. Since $(g, h, \tilde{g}, \tilde{h})$ is not a DH-tuple, there exist $a, b \in \mathbb{Z}_q$ and $a \neq b$, such that $\tilde{g} = g^a$ and $\tilde{h} = h^b$. We show that $Pr[RAND(g, h, \tilde{g}, \tilde{h}) = (g^\alpha, g^\beta)] = \frac{1}{q^2}$ where the probability is taken over s, t used to compute $RAND$.
Let $\gamma \in \mathbb{Z}_q$ be such that $h = g^\gamma$, then $(u, v) \leftarrow RAND(g, h, \tilde{g}, \tilde{h}) = RAND(g, g^\gamma, g^a, (g^\gamma)^b)$, so $u = g^s \cdot (g^\gamma)^t = g^{s+\gamma \cdot t}$ and $v = (g^a)^s \cdot (g^{\gamma \cdot b})^t = g^{a \cdot s + \gamma \cdot b \cdot t}$. Thus $(u, v) = (g^\alpha, g^\beta)$ if and only if

$$\begin{cases} s + \gamma \cdot t = \alpha \\ a \cdot s + \gamma \cdot b \cdot t = \beta \end{cases}$$

These equations have a single solution if and only if the matrix

$$\begin{pmatrix} 1 & \gamma \\ a & \gamma b \end{pmatrix}$$

is invertible, which holds here since the determinant is $1 \cdot \gamma b - \gamma \cdot a = \gamma(b - a) \neq 0$, where the inequality holds since $a \neq b$. Thus, there is a single pair s, t such that $(u, v) = (g^\alpha, g^\beta)$ implying that the probability is $\frac{1}{q^2}$, as required.

2.3 An Encryption Scheme Based on *RAND*

Based on algorithm $RAND$, Peikert et al. construct a public key cryptosystem in [12], whose correctness and security are based on Proposition 1 and Proposition 2.

Scheme 1: Basic Public-key Encryption Scheme Based on $RAND$: *Encryption*1.

- **KeyGen(1^n):** Choose $(\mathbb{G}, g_0, q)$ on the security parameter 1^n, where $\mathbb{G}$ is a group of prime order q with generator g_0, and q is of length n. The message space of the system is $\mathbb{G}$. Randomly choose $g, h \in \mathbb{G}$ and exponent $x \in \mathbb{Z}_q$. Let $pk = (g, h, g^x, h^x)$ and $sk = x$.
- **Enc(pk, m):** Parse pk as $(g, h, \tilde{g}, \tilde{h})$. Let $(u, v) \leftarrow RAND(g, h, \tilde{g}, \tilde{h})$. Output the ciphertext $(u, v \cdot m)$.
- **Dec(sk, c):** Parse c as (c_0, c_1). Output $m = c_1 / c_0^{sk} = c_1 / c_0^x$.

Notice that if we set a DH-tuple as public key in *Encryption*1, then the ciphertext can be decrypted correctly. Otherwise, if we set a Non-DH-tuple as public key, then we can only get a random element in $\mathbb{G}$.

3 Cut-and-choose OT Protocol

In [5], Lindell et al. proposed their cut-and-choose OT protocol, and then modified it in [6]. In this section, we extract a simplified version of their protocol, and describe it in a different way.

3.1 The Functionality of Cut-and-choose OT Protocol

Functionality: $\mathcal{F}_{\mathcal{CACOT}}$

- **Inputs:** The sender's input is a pair (k_0, k_1), the receiver's inputs are bits σ and j.
- **Auxiliary input:** Both parties hold a security parameter 1^n and $(\mathbb{G}, g_0, q)$, where $\mathbb{G}$ is a group of order q with a generator g_0, and q is of length n.

– **Output:**
  - The sender outputs nothing;
  - The receiver outputs that
    if $j = 0$ **then** outputs k_0, k_1;
    else if $j = 1$ **then** outputs k_σ;

3.2 First Variant of $\boldsymbol{RAND}$: $\boldsymbol{ExtenedRAND}$

The output of $RAND$ is one pair and can be used to encrypt only one message. In the OT protocol, we must encrypt two messages at the same time. So we construct an algorithm based on $RAND$ whose outputs are two pairs, which is called $ExtenedRAND$.

Algorithm 2: $ExtenedRAND(g, h, g_1, h_1, \tilde{g}, \tilde{h})$

Let $(\mathbb{G}, g_0, q)$ be such that $\mathbb{G}$ is a group of order q, with generator g_0. Let $g, h, g_1, h_1, \tilde{g}, \tilde{h} \in \mathbb{G}$. Computes

1. $(u_0, v_0) \leftarrow RAND(g, h, \tilde{g}, \tilde{h})$
2. $(u_1, v_1) \leftarrow RAND(g_1, h_1, \tilde{g}, \tilde{h})$

Output $((u_0, v_0), (u_1, v_1))$.

3.3 Cut-and-choose OT from $\boldsymbol{ExtenedRAND}$

Protocol: We describe the cut-and-choose OT protocol as follows:

Protocol 1: Cut-and-choose OT From $ExtenedRAND$: $OT1$.

1. The receiver R chooses randomly $h_0 \in \mathbb{G}$ and $a \in \mathbb{Z}_q$, computes $g_1 = (g_0)^a$, $h_1 = (h_0)^{a+j}$, that is
   if $j = 0$ **then** computes $h_1 = (h_0)^a$;
   else if $j = 1$ **then** computes $h_1 = (h_0)^{a+1}$;
2. P sends (h_0, g_1, h_1) to S.
3. R proves to S that he know the discrete logarithm of g_1 relative to g_0.
4. R chooses a random $b \in \mathbb{Z}_q$, and computes $\tilde{g} = (g_\sigma)^b, \tilde{h} = (h_\sigma^b)$, that is
   if $\sigma = 0$ **then** computes $\tilde{g} = g_0^b, \tilde{h} = h_0^b$;
   else if $\sigma = 1$ **then** computes $\tilde{g} = g_1^b, \tilde{h} = h_1^b$;
5. S computes $((u_0, v_0), (u_1, v_1)) \leftarrow ExtenedRAND(g_0, h_0, g_1, h_1, \tilde{g}, \tilde{h})$, and $w_0 = v_0 \cdot k_0$, $w_1 = v_1 \cdot k_1$.
6. S sends $((u_0, w_0), (u_1, w_1))$ to R.
7. R computes that
   if $j = 0$ **then**
      if $\sigma = 0$ **then** computes $k_0 = w_0/(u_0)^b$, $k_1 = w_1/(u_1)^{ba^{-1}}$;
      else if $\sigma = 1$ **then** computes $k_0 = w_0/(u_0)^{ab}$, $k_1 = w_1/(u_1)^b$;
   else if $j = 1$ **then**
      if $\sigma = 0$ **then** computes $k_0 = w_0/(u_0)^b$;
      else if $\sigma = 1$ **then** computes $k_1 = w_1/(u_1)^b$;

Correctness: We summarize all cases of the combination of j and σ's values. The correctness of the protocol is easy to see from Table 1.

Table 1. The running branches of Protocol 1

(j,σ)	Input: g,h g_1,h_1 $\tilde{g},\tilde{h}$	Output of $ExtenedRAND(g,h,g_1,h_1,\tilde{g},\tilde{h})$	Properties
$(0,0)$	g_0,h_0 g_0^a,h_0^a $(g_0)^b,(h_0)^b$	$-\ (u_0,v_0)\leftarrow RAND(g_0,h_0,(g_0)^b,(h_0)^b)$ $-\ (u_1,v_1)\leftarrow RAND(g_0^a,h_0^a,g_0^b,h_0^b)$ $= RAND(g_0^a,h_0^a,(g_0^a)^{ba^{-1}},(h_0^a)^{ba^{-1}})$	$-\ v_0=u_0^b$ $-\ v_1=u_1^{ba^{-1}}$
$(0,1)$	g_0,h_0 g_0^a,h_0^a $(g_0^a)^b,(h_0^a)^b$	$-\ (u_0,v_0)\leftarrow RAND(g_0,h_0,(g_0^a)^b,(h_0^a)^b)$ $= RAND(g_0,h_0,(g_0)^{ab},(h_0)^{ab})$ $-\ (u_1,v_1)\leftarrow RAND(g_0^a,h_0^a,(g_0^a)^b,(h_0^a)^b)$	$-\ v_0=u_0^{ba}$ $-\ v_1=u_1^b$
$(1,0)$	g_0,h_0 g_0^a,h_0^{a+1} $(g_0)^b,(h_0)^b$	$-\ (u_0,v_0)\leftarrow RAND(g_0,h_0,(g_0)^b,(h_0)^b)$ $-\ (u_1,v_1)\leftarrow RAND(g_0^a,h_0^{a+1},g_0^b,h_0^b)$	$-\ v_0=u_0^b$ $-\ (\ldots,u_1,v_1)\overset{c}{\approx}(\ldots,g_0^\alpha,g_0^\beta)$ where $\alpha,\beta\in_R\mathbb{Z}_q$
$(1,1)$	g_0,h_0 g_0^a,h_0^{a+1} $(g_0^a)^b,(h_0^{a+1})^b$	$-\ (u_0,v_0)\leftarrow RAND(g_0,h_0,(g_0)^b,(h_0^{a+1})^b)$ $-\ (u_1,v_1)\leftarrow RAND(g_0^a,h_0^{a+1},g_0^b,(h_0^{a+1})^b)$	$-\ (\ldots,u_0,v_0)\overset{c}{\approx}(\ldots,g_0^\alpha,g_0^\beta)$ where $\alpha,\beta\in_R\mathbb{Z}_q$. $-\ v_1=u_1^b$

Security: The security proof is as same as the proof in [6], so we omit it here.

Efficiency: To construct the inputs of $ExtenedRAND$ (the inputs can be viewed as a public key to the encryption scheme based on $RAND$), R computes 4 exponentiations; to construct the ciphertext, S computes 8 exponentiations and 6 multiplications; to decrypt the ciphertext, if $j=0$, R computes 2 exponentiations, 2 modular inverses and 2 multiplications, if $j=1$, R computes 1 exponentiation, 1 modular inverse and 1 multiplication, the expectation of computes are 1.5 exponentiations, 1.5 modular inverses and 1.5 multiplications. The protocol takes 3 rounds of communication, and the parties exchange 9 group elements (5 of them are elements of the public key, and 4 of them are ciphertexts). It also needs 1 zero knowledge proof of discrete logarithm.

4 Cut-and-choose Inverse OT Protocol

4.1 The Functionality of Cut-and-choose Inverse OT Protocol

Functionality: $\mathcal{F}_{\mathcal{CACIOT}}$

- **Inputs:** The sender S's inputs are a key-pair (k_0,k_1), a bit τ and a bit m, m indicates whether the order of the keys is permuted; the receiver R's input is a bit j.

- **Auxiliary input:** Both parties hold a security parameter 1^n and $(\mathbb{G}, g_0, q)$, where $\mathbb{G}$ is a group of order q with a generator g_0, and q is of length n. The commitments $com(k_m), com(k_{1-m}), com(m)$ to (k_m, k_{1-m}) and m, that is:
  if $m = 0$ **then** the commitments are $com(k_0), com(k_1), com(m)$;
  else if $m = 1$ **then** the commitments are $com(k_1), com(k_0), com(m)$;
- **Output:**
  - The sender outputs nothing;
  - The receiver outputs that
    if $j = 0$ **then** outputs k_0, k_1 and m;
    else if $j = 1$ **then** outputs k_τ;

In order to realize $\mathcal{F}_{\mathcal{CACIOT}}$, the following four aspects must be paid attention to.

- From the definition of $\mathcal{F}_{\mathcal{CACIOT}}$, we observe that the key k_τ corresponding to the sender's input bit τ is always obtained by the receiver, no matter with the value of j, but the other key $k_{1-\tau}$ and m are obtained by the receiver only when $j = 0$. Just as in protocol $OT1$, R sends S a DH-tuple when $j = 0$ and a Non-DH-tuple when $j = 1$, this tuple is suitable for encrypting $k_{1-\tau}$ and m, but not for k_τ (because when $j = 1$, R can not get k_τ). So we make S construct a DH-tuple himself based on the tuple received from R. When receiving $(g_0, g_0^a, \tilde{g}, \tilde{h})$ from R, S can randomly select $r \in \mathbb{Z}_q$, construct $(g_0, g_0^r, g_0^a, (g_0^a)^r)$ as a DH-tuple, and use it to encrypt k_τ. R can then decrypt the ciphertext using a.
- We must randomly permute the order of the k_0 and k_1, otherwise, it will leak the input τ. For example, if $j = 1$ and R gets the first key, then R knows that $\tau = 0$. We permute the order of the keys by a random bit m, if $m = 0$, we keep the order; if $m = 1$, we change the order.
- Compared with $j = 0$, the situation is more complicate in the case of $j = 1$. In this case, R does not know S's input τ, and the key order perhaps be changed, so he is not aware of which ciphertext should be decrypted. To solve this problem, we use the approach of committed OT. S computes $com(k_0), com(k_1), com(m)$, randomly changes their order and transfers them to R before OT is executed, so those commitments can be viewed as auxiliary input of the OT. Then R can know which one is k_τ by verifying the auxiliary commitments.
- We must use the same decryption key to decrypt all of k_τ, $k_{1-\tau}$ and m, otherwise, it will leak the input τ. For example, when $j = 0$, R can get both k_0, k_1 and m, because m and $k_{1-\tau}$ are encrypted by same key and k_τ is decrypted by the other key, we can get τ by this difference.

4.2 Second Variant of $\boldsymbol{RAND}$: $\boldsymbol{SelfExtendedRAND}$

We construct an algorithm $ShrinkedRAND$ based on $RAND$, which forms a DH-tuple from a DH-pair firstly, and then calls the algorithm $RAND$.

Algorithm 3: $ShrinkedRAND(g, h)$

Let $(\mathbb{G}, g_0, q)$ be such that $\mathbb{G}$ is a group of order q, with generator g_0. Elements $g, h \in \mathbb{G}$. Choose $r \leftarrow \mathbb{Z}_q$, computes $\tilde{g} = g^r$ and $\tilde{h} = h^r$ and

$$(u, v) \leftarrow RAND(g, \tilde{g}, h, \tilde{h})$$

Output (u, v).

Algorithm $SelfExtendedRAND$'s input is as same as $RAND$, but outputs three pairs, which can be used to encrypt three messages $k_\tau, k_{1-\tau}, m$.

Algorithm 4: $SelfExtendedRAND(g, h, g_1, h_1)$

Let $(\mathbb{G}, g_0, q)$ be such that $\mathbb{G}$ is a group of order q, with generator g_0. Elements $g, h, g_1, g_2 \in \mathbb{G}$.
Computes

1. $(u_0, v_0) \leftarrow ShrinkedRAND(g, h)$
2. $(u_1, v_1) \leftarrow RAND(g, g_1, h, h_1)$
3. $(u_2, v_2) \leftarrow RAND(g, g_1, h, h_1)$

Output $((u_0, v_0), (u_1, v_1), (u_2, v_2))$.

Note 1. (u_1, v_1) and (u_2, v_2) are generated by $RAND$ with same inputs, but different random numbers, and the order of inputs to $RAND$ is different to the inputs order in $SelfExtendedRAND$, which can make $(u_0, v_0), (u_1, v_1), (u_2, v_2)$ be DH-pairs with same exponent.

4.3 Cut-and-choose Inverse OT from $SelfExtenedRAND$

Protocol: We describe the cut-and-choose inverse OT protocol as follows:

Protocol 2: Cut-and-choose Inverse OT From $SelfExtenedRAND$: $OT2$.

1. The receiver R chooses randomly $a, b \in \mathbb{Z}_q$, computes $h_0 = (g_0)^a$,$g_1 = (g_0)^b$, $h_1 = (h_0)^{b+j}$, that is
   if $j = 0$ **then** computes $h_1 = (h_0)^b = ((g_0)^a)^b$;
   else if $j = 1$ **then** computes $h_1 = (h_0)^{b+1} = ((g_0)^a)^{b+1}$;
2. P sends (h_0, g_1, h_1) to S.
3. R proves to S that he know the discrete logarithm of g_1 relative to g_0.
4. S computes $((u_0, v_0), (u_1, v_1), (u_2, v_2)) \leftarrow SelfExtenedRAND(g_0, h_0, g_1, h_1)$, and $w_0 = v_0 \cdot k_\tau$, $w_1 = v_1 \cdot k_{1-\tau}$, $w_2 = v_2 \cdot m$.
5. S permutes the order of $(u_0, w_0), (u_1, w_1)$ randomly as $(u_0', w_0'), (u_1', w_1')$, sends $((u_0', w_0'), (u_1', w_1'), (u_2, w_2))$ to R.
6. R computes $d_0 = w_0'/(u_0')^a$,$d_1 = w_1'/(u_1')^a$.
   - If $j = 0$, R computes $m = w_2/(u_2)^a$, verifies if $com(m)$ is the commitment of m, $com(k_m)$ is the commitment of one of d_0, d_1, and $com(k_{1-m})$ is the commitment of the other one. If not, R outputs $\perp$, otherwise, he can rearrange the order of $com(k_m), com(k_{1-m})$ as $com(k_0), com(k_1)$, and then rearrange the order of d_0 and d_1 as k_0, k_1.
   - If $j = 1$, R verifies whether only one of $com(k_m), com(k_{1-m})$ is the commitment of one of d_0 and d_1, if not, R outputs $\perp$, otherwise it outputs the one in d_0 and d_1, which is correctly verified as k_τ.

Note 2. In the protocol, the encryption algorithm is not suitable to encrypt a single bit, so it can not encrypt m directly. This problem can be solved by message padding, we omit the details here.

Correctness: The correctness of the protocol is easy to see from Table2.

Table 2. The running branches of Protocol 2

j	Input: g, h, g_1, h_1	Outputs of $SelfExtenedRAND$:(g, h, g_1, h_1)	Properties
0	$g_0, g_0^a, g_0^b, (g_0^a)^b$	$-$ $(u_0, v_0) \leftarrow RAND(g_0, g_0^r, g_0^a, (g_0^r)^a)$ where $r \in_R \mathbb{Z}_q$ $-$ $(u_1, v_1) \leftarrow RAND(g_0, g_0^b, g_0^a, (g_0^a)^b)$ $-$ $(u_2, v_2) \leftarrow RAND(g_0, g_0^b, g_0^a, (g_0^a)^b)$	$-$ $v_0 = u_0^a$ (encrypt k_τ) $-$ $v_1 = u_1^a$ (encrypt $k_{1-\tau}$) $-$ $v_2 = u_2^a$ (encrypt m)
1	$g_0, g_0^a, g_0^b, (g_0^a)^{b+1}$	$-$ $(u_0, v_0) \leftarrow RAND(g_0, g_0^r, g_0^a, (g_0^r)^a)$ where $r \in_R \mathbb{Z}_q$ $-$ $(u_1, v_1) \leftarrow RAND(g_0, g_0^b, g_0^a, (g_0^a)^{b+1})$ $-$ $(u_2, v_2) \leftarrow RAND(g_0, g_0^b, g_0^a, (g_0^a)^{b+1})$	$-$ $v_0 = u_0^a$ (encrypt k_τ) $-$ $(\dots, u_1, v_1) \overset{c}{\approx} (\dots, g_0^\alpha, g_0^\beta)$ where $\alpha, \beta \in_R \mathbb{Z}_q$ (encrypt $k_{1-\tau}$) $-$ $(\dots, u_2, v_2) \overset{c}{\approx} (\dots, g_0^{\alpha'}, g_0^{\beta'})$ where $\alpha', \beta' \in_R \mathbb{Z}_q$ (encrypt m)

Security: The security of the protocol is proved by following theorem.

Theorem 1. *If the Decisional Diffie-Hellman assumption holds in group $\mathbb{G}$, then the protocol 1 securely computes $\mathcal{F}_{\mathcal{CACIOT}}$ functionality in the presence of malicious adversaries.*

Proof. We prove security in a hybrid model where the zero-knowledge proofs and proofs of knowledge (ZKPOK) are computed by ideal functionalities.

R **is corrupted:** Let $\mathcal{A}$ be an adversary that controls the receiver R in real world. We construct a simulator $\mathcal{S}$ that invokes $\mathcal{A}$ on its input and works as follows:

1. $\mathcal{S}$ receives (h_0, g_1, h_1) from $\mathcal{A}$ and verifies the zero-knowledge proof as the honest sender would.
   (a) If the verification fails, $\mathcal{S}$ sends $\perp$ to the trusted party and halts.
   (b) Otherwise, $\mathcal{S}$ runs the extractor and extracts a witness α. Then $\mathcal{S}$ sets $j = 0$ if $(h_0)^\alpha = h_1$ and $j = 1$ otherwise.
2. The simulator $\mathcal{S}$ sends j to the trusted party:
   (a) If $j = 0$, $\mathcal{S}$ receives back k_0, k_1 and m.
   (b) If $j = 1$, $\mathcal{S}$ receives back k_τ.
3. Like the honest sender, $\mathcal{S}$ simulates the transfer of k_0, k_1, m as follows: $\mathcal{S}$ computes
$$((u_0, v_0), (u_1, v_1), (u_2, v_2)) \leftarrow SelfExtenedRAND(g_0, h_0, g_1, h_1),$$
   and $w_0 = v_0 \cdot k_\tau$, $w_1 = v_1 \cdot k_{1-\tau}$, $w_2 = v_2 \cdot m$, and permutes the order of $(u_0, w_0), (u_1, w_1)$ randomly as $(u_0', w_0'), (u_1', w_1')$.
4. $\mathcal{S}$ sends $(u_0', w_0'), (u_1', w_1'), (u_2, w_2)$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

It is easy to see that the outputs of the ideal execution between $\mathcal{S}$ and an honest sender S is identical to the outputs of a real execution with $\mathcal{A}$ and an honest sender S.

S **is corrupted:** We now consider the case that $\mathcal{A}$ controls S. We construct a simulator $\mathcal{S}$ that invokes $\mathcal{A}$ on its inputs and works as follows:

1. $\mathcal{S}$ chooses random $a, b \in \mathbb{Z}_q$ and computes $h_0 = (g_0)^a, g_1 = (g_0)^b$. Then it sets $h_1 = (g_1)^a$ which means that $j = 0$, and sends (h_0, g_1, h_1) to $\mathcal{A}$.
2. $\mathcal{S}$ proves to $\mathcal{A}$ that he knows the discrete logarithm of h_0 relative to g_0.
3. $\mathcal{S}$ receives $(u_0', w_0'), (u_1', w_1'), (u_2, w_2)$ from $\mathcal{A}$, like an honest R, $\mathcal{S}$ can decrypts $(u_0', w_0'), (u_1', w_1'), (u_2, w_2)$ and verifies the commitments. If there exists a commitment that does not hold, the $\mathcal{S}$ outputs $\perp$, otherwise, because in this case $j = 0$, $\mathcal{S}$ can get k_0, k_1, m.
4. $\mathcal{S}$ rewinds to the first step, and sets $h_1 = (g_1)^{a+1}$, which means that $j = 1$.
5. $\mathcal{S}$ interacts with $\mathcal{A}$ as an honest R, if $\mathcal{S}$ outputs $\perp$, then $\mathcal{S}$ rewinds to step 1 just like step4. Repeats this process until $\mathcal{S}$ extracts k_τ.
6. $\mathcal{S}$ compares k_τ with k_0, k_1, then he can get τ.
7. $\mathcal{S}$ sends k_0, k_1, m, τ to the trusted party, outputs whatever $\mathcal{A}$ outputs and halts.

The main observation of the simulation is how to extract the input of τ. The simulator firstly sets $j = 0$ and obtains values (k_0, k_1, m) (by m, it can get the correct order of k_0 and k_1), then it rewinds to the first step by setting $j = 1$ again and again until it gets k_τ. Just like discussion in [4] and [5], the rewind process can be finished in expected polynomial time. Comparing k_τ with k_0 and k_1, then the simulator gets τ. It is easy to see that the outputs of the ideal execution between $\mathcal{S}$ and an honest receiver R is identical to the outputs of a real execution with $\mathcal{A}$ and an honest honest receiver R.

Efficency: To construct the inputs of $SelfExtenedRAND$ (the inputs can be view as a public key to encryption scheme based on $RAND$), R computes 3 exponentiations; to construct the ciphertext, S computes 14 exponentiations and 8 multiplications; to decrypt the ciphertext, if $j = 0$, R computes 3 exponentiations, 3 modular inverses, 3 multiplications and 3 commitments verifications, if $j = 1$, R computes 2 exponentiations, 2 modular inverses, 2 multiplications and 2 commitments verifications, the expectation of computes are 2.5 exponentiations, 2.5 modular inverses, 2.5 multiplications and 2.5 commitments verifications. The protocol takes 3 rounds of communications, and the parties exchange 9 group elements (3 of them are elements of the public key, and 6 of them are ciphertexts). It also need 1 zero knowledge proof of discrete logarithm.

About Selective Failure Attack We do not specify the commitment scheme used in the protocol. If we adapt the method of commited OT that is discussed in [11], it can ensure our scheme against the selective failure attack. This method is also taken in [5] and [6], which used a DDH type commitment with a proof of knowledge of discrete logarithm.

5 Cut-and-choose Bilateral Oblivious Transfer

5.1 The Functionality of Cut-and-choose Bilateral OT Protocol

Functionality: $\mathcal{F}_{\mathcal{CACBOT}}$

– **Inputs:** The sender S's inputs are a a bit τ and $(k_0^1, k_1^1, m), (k_0^2, k_1^2)$, which m indicates the order of the key (k_0^1, k_1^1); the receiver R's input are bit j and σ.

– **Auxiliary input:** Both parties hold a security parameter 1^n and $(\mathbb{G}, g_0, q)$, where $\mathbb{G}$ is a group of order q with a generator g_0, and q is of length n. The commitments $com(k_m^1), com(k_{1-m}^1), com(m)$ to (k_m, k_{1-m}) and m, that is
if $m = 0$ **then** the commitments are $com(k_0^1), com(k_1^1), com(m)$;

else if $m = 1$ **then** the commitments are $com(k_1^1), com(k_0^1), com(m)$;

– **Output:**

- The sender outputs nothing
- The receiver outputs that
  if $j = 0$ **then** outputs $(k_0^1, k_1^1, k_0^2, k_1^2)$ and m;

  else if $j = 1$ **then** outputs (k_τ, k_σ);

5.2 A Combination of *ExtendedRAND* and *SelfExtendedRAND*: *CombinedRAND*

Combining the $ExtendedRAND$ and $SelfExtendedRAND$, we form the $CombinedRAND$, it outputs five pairs that can encrypt five messages.

Algorithm 5: $CombinedRAND(g, h, g_1, h_1, \tilde{g}, \tilde{h})$

Let $(\mathbb{G}, g_0, q)$ be such that $\mathbb{G}$ is a group of order q, with generator g_0. Elements $g, h, g_1, h_1, \tilde{g}, \tilde{h} \in \mathbb{G}$.
Computes

1. $((u_0, v_0), (u_1, v_1), (u_2, v_2)) \leftarrow SelfExtendedRAND(g, h, g_1, h_1)$
2. $((u_3, v_3), (u_4, v_4)) \leftarrow ExtenedRAND(g, h, g_1, h_1, \tilde{g}, \tilde{h})$

Output $((u_0, v_0), (u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4))$.

5.3 Cut-and-choose Bilateral OT from *CombinedRAND*

Protocol: We describe the cut-and-choose bilateral OT protocol as follows:

Protocol 3: Cut-and-choose Bilateral OT From $CombinedRAND$: $OT3$.

1. The receiver R chooses randomly $a, b \in \mathbb{Z}_q$, computes $h_0 = (g_0)^a$,$g_1 = (g_0)^b$, $h_1 = (h_0)^{b+j}$, that is
   if $j = 0$ **then** computes $h_1 = (h_0)^b = ((g_0)^a)^b$;
   else if $j = 1$ **then** computes $h_1 = (h_0)^{b+1} = ((g_0)^a)^{b+1}$;
2. P sends (h_0, g_1, h_1) to S.
3. R proves to S that he knows the discrete logarithm of g_1 relative to g_0.
4. R chooses a random $c \in \mathbb{Z}_q$, and computes $\tilde{g} = (g_\sigma)^c, \tilde{h} = (h_\sigma^c)$, that is
   if $\sigma = 0$ **then** computes $\tilde{g} = g_0^c, \tilde{h} = h_0^c$;
   else if $\sigma = 1$ **then** computes $\tilde{g} = g_1^c, \tilde{h} = h_1^c$;
5. S computes $((u_0, v_0), (u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4)) \leftarrow CombinedRAND(g_0, h_0, g_1, h_1, \tilde{g}, \tilde{h})$, and $w_0 = v_0 \cdot k_\tau^1$, $w_1 = v_1 \cdot k_{1-\tau}^1$, $w_2 = v_2 \cdot m$, $w_3 = v_3 \cdot k_0^2$, $w_4 = v_4 \cdot k_1^2$.
6. S randomly permutes the order of $(u_0, w_0), (u_1, w_1)$ as $(u_0', w_0'), (u_1', w_1')$, sends $((u_0', w_0'), (u_1', w_1'), (u_2, w_2)(u_3, w_3), (u_4, w_4))$ to R.
7. R computes $d_0 = w_0'/(u_0')^a$, $d_1 = w_1'/(u_1')^a$.
   - If $j = 0$,
     - R computes $m = w_3/(u_3)^a$, verifies if $com(m)$ is the commitment of m, $com(k_m^1)$ is the commitment of one of d_0, d_1, and $com(k_{1-m}^1)$ is the commitment of the other one. If not, R outputs $\perp$, otherwise, he can rearrange the order of $com(k_m^1), com(k_{1-m}^1)$ as $com(k_0^1), com(k_1^1)$, and then rearrange the order of d_0 and d_1 as k_0^1, k_1^1.
     - **if** $\sigma = 0$ **then** computes $k_0^2 = w_3/(u_3)^c$, $k_1^2 = w_4/(u_4)^{cb^{-1}}$;
     - **else if** $\sigma = 1$ **then** computes $k_0^2 = w_3/(u_3)^{bc}$, $k_1^2 = w_4/(u_4)^c$;
   - If $j = 1$,
     - R verifies whether only one of $com(k_m^1), com(k_{1-m}^1)$ is the commitment of one of d_0 and d_1, if not, R outputs $\perp$, otherwise it outputs the one in d_0 and d_1, which is correctly verified as k_τ.
     - **if** $\sigma = 0$ **then** computes $k_0^2 = w_3/(u_3)^c$;
     - **else if** $\sigma = 1$ **then** computes $k_1^2 = w_4/(u_4)^c$;

Correctness: The correctness of the protocol is easy to see from Table3.

Table 3. The running branches of Protocol 3

(j, σ)	Input: g, h g_1, h_1 $\tilde{g}, \tilde{h}$	Output of $CombinedRAND(g, h, g_1, h_1, \tilde{g}, \tilde{h})$	Properties
$(0, 0)$	g_0, g_0^a $(g_0)^b, (g_0^a)^b$ $(g_0)^c, (g_0^a)^c$	– $(u_0, v_0) \leftarrow RAND(g_0, g_0^r, g_0^a, (g_0^r)^a)$ where $r \in_R \mathbb{Z}_q$ – $(u_1, v_1) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^b)$ – $(u_2, v_2) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^b)$ – $(u_3, v_3) \leftarrow RAND(g_0, g_0^a, (g_0)^c, (g_0^a)^c)$ – $(u_4, v_4) \leftarrow RAND((g_0)^b, (g_0^a)^b, (g_0)^c, (g_0^a)^c)$	– $v_0 = u_0^a$ (encrypt k_τ^1) – $v_1 = u_1^a$ (encrypt $k_{1-\tau}^1$) – $v_2 = u_2^a$ (encrypt m) – $v_3 = u_3^c$ (encrypt k_0^2) – $v_4 = u_4^{cb^{-1}}$ (encrypt k_1^2)

$(0,1)$	g_0, g_0^a $(g_0)^b, (g_0^a)^b$ $(g_0^b)^c, ((g_0^a)^b)^c$	$-$ $(u_0, v_0) \leftarrow RAND(g_0, g_0^r, g_0^a, (g_0^r)^a)$ where $r \in_R \mathbb{Z}_q$ $-$ $(u_1, v_1) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^b)$ $-$ $(u_2, v_2) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^b)$ $-$ $(u_3, v_3) \leftarrow RAND(g_0, g_0^a, (g_0^b)^c, ((g_0^a)^b)^c)$ $-$ $(u_4, v_4) \leftarrow RAND((g_0)^b, (g_0^a)^b, (g_0^b)^c, ((g_0^a)^b)^c)$	$-$ $v_0 = u_0^a$ (encrypt k_τ^1) $-$ $v_1 = u_1^a$ (encrypt $k_{1-\tau}^1$) $-$ $v_2 = u_2^a$ (encrypt m) $-$ $v_3 = u_3^{bc}$ (encrypt k_0^2) $-$ $v_4 = u_4^c$ (encrypt k_1^2)
$(1,0)$	g_0, g_0^a $g_0^b, (g_0^a)^{b+1}$ $(g_0)^c, (g_0^a)^c$	$-$ $(u_0, v_0) \leftarrow RAND(g_0, g_0^r, g_0^a, (g_0^r)^a)$ where $r \in_R \mathbb{Z}_q$ $-$ $(u_1, v_1) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^{b+1})$ $-$ $(u_2, v_2) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^{b+1})$ $-$ $(u_3, v_3) \leftarrow RAND(g_0, g_0^a, (g_0)^c, (g_0^a)^c)$ $-$ $(u_4, v_4) \leftarrow RAND((g_0)^b, (g_0^a)^{b+1}, (g_0)^c, (g_0^a)^c)$	$-$ $v_0 = u_0^a$ (encrypt k_τ^1) $-$ $(\ldots, u_1, v_1) \overset{c}{\approx} (\ldots, g_0^\alpha, g_0^\beta)$ (encrypt $k_{1-\tau}^1$) $-$ $(\ldots, u_2, v_2) \overset{c}{\approx} (\ldots, g_0^{\alpha'}, g_0^{\beta'})$ (encrypt m) $-$ $v_3 = u_3^c$ (encrypt k_0^2) $-$ $(\ldots, u_4, v_4) \overset{c}{\approx} (\ldots, g_0^{\alpha''}, g_0^{\beta''})$ (encrypt k_1^2)
$(1,1)$	g_0, g_0^a $g_0^b, (g_0^a)^{b+1}$ $(g_0^b)^c, ((g_0^a)^{b+1})^c$	$-$ $(u_0, v_0) \leftarrow RAND(g_0, g_0^r, g_0^a, (g_0^r)^a)$ where $r \in_R \mathbb{Z}_q$ $-$ $(u_1, v_1) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^{b+1})$ $-$ $(u_2, v_2) \leftarrow RAND(g_0, (g_0)^b, g_0^a, (g_0^a)^{b+1})$ $-$ $(u_3, v_3) \leftarrow RAND(g_0, g_0^a, ((g_0)^b)^c, ((g_0^a)^{b+1})^c)$ $-$ $(u_4, v_4) \leftarrow RAND((g_0)^b, (g_0^a)^{b+1},$ $((g_0)^b)^c, ((g_0^a)^{b+1})^c)$	$-$ $v_0 = u_0^a$ (encrypt k_τ^1) $-$ $(\ldots, u_1, v_1) \overset{c}{\approx} (\ldots, g_0^\alpha, g_0^\beta)$ (encrypt $k_{1-\tau}^1$) $-$ $(\ldots, u_2, v_2) \overset{c}{\approx} (\ldots, g_0^{\alpha'}, g_0^{\beta'})$ (encrypt m) $-$ $(\ldots, u_3, v_3) \overset{c}{\approx} (\ldots, g_0^{\alpha''}, g_0^{\beta''})$ (encrypt k_0^2) $-$ $v_4 = u_4^c$ (encrypt k_1^2)

In the table, $\alpha, \beta, \alpha', \beta', \alpha'', \beta'' \in_R \mathbb{Z}_q$

Security: The security of the protocol is proved by following theorem.

Theorem 2. *If the Decisional Diffie-Hellman assumption holds in group $\mathbb{G}$, then the protocol 3 securely computes $\mathcal{F}_{\mathcal{CACBOT}}$ functionality in the presence of malicious adversaries.*

Proof. We prove security in a hybrid model where the zero-knowledge proofs and proofs of knowledge (ZKPOK) are computed by ideal functionalities.

R **is corrupted:** Let $\mathcal{A}$ be an adversary that controls the receiver R in real word. We construct a simulator $\mathcal{S}$ that invokes $\mathcal{A}$ on its input and works as follows:

1. $\mathcal{S}$ receives (h_0, g_1, h_1) from $\mathcal{A}$ and verifies the zero-knowledge proof as the honest sender would.
   (a) If the verification fails, $\mathcal{S}$ sends $\perp$ to the trusted party and halts.
   (b) Otherwise, $\mathcal{S}$ runs the extractor and extracts a witness α. Then $\mathcal{S}$ sets $j = 0$ if $(g_1)^\alpha = h_1$ and $j = 1$ otherwise.

2. $\mathcal{S}$ receives $(\tilde{g}, \tilde{h})$ from $\mathcal{A}$
3. If $j = 0$, $\mathcal{S}$ set $\sigma = 0$ if $(\tilde{g})^{\alpha} = \tilde{h}$ and $\sigma = 1$ otherwise.
4. If $j = 1$, $\mathcal{S}$ set σ arbitrarily, say to equal 0.
5. The simulator $\mathcal{S}$ sends σ, j to the trusted party:
   (a) If $j = 0$, $\mathcal{S}$ receives back $k_0^1, k_1^1, m, k_0^2, k_1^2$.
   (b) If $j = 1$, $\mathcal{S}$ receives back k_τ^1, k_σ^2.
6. Like the honest sender, $\mathcal{S}$ simulates the transfer of $k_0^1, k_1^1, m, k_0^2, k_1^2$ as follows: $\mathcal{S}$ computes $((u_0, v_0), (u_1, v_1)), (u_2, v_2), (u_3, v_3), (u_4, v_4)) \leftarrow CombinedRAND(g_0, h_0, g_1, h_1, \tilde{g}, \tilde{h})$, and $w_0 = v_0 \cdot k_\tau^1$, $w_1 = v_1 \cdot k_{1-\tau}^1$, $w_2 = v_2 \cdot m$, $w_3 = v_3 \cdot k_0^2$, $w_4 = v_4 \cdot k_1^2$ and randomly permutes the order of $(u_0, w_0), (u_1, w_1)$ as $(u_0', w_0'), (u_1', w_1')$.
7. $\mathcal{S}$ sends $(u_0', w_0'), (u_1', w_1'), (u_2, w_2), (u_3, w_3), (u_4, w_4)$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

It is easy to see that the outputs of the ideal execution between $\mathcal{S}$ and an honest sender S is identical to the outputs of a real execution with $\mathcal{A}$ and an honest sender S.

S **is corrupted:** We now consider the case that $\mathcal{A}$ controls S. We construct a simulator $\mathcal{S}$ that invokes $\mathcal{A}$ on its inputs and works as follows:

1. $\mathcal{S}$ chooses random $a, b \in \mathbb{Z}_q$ and computes $h_0 = (g_0)^a, g_1 = (g_0)^b$. Then it sets $h_1 = (g_1)^a$ which means that $j = 0$, and sends (h_0, g_1, h_1) to $\mathcal{A}$.
2. $\mathcal{S}$ proves to $\mathcal{A}$ that he know the discrete logarithm of h_0 relative to g_0.
3. $\mathcal{S}$ chooses random $c \in \mathbb{Z}_q$, computes $\tilde{g} = (g_\sigma)^c, \tilde{h} = (h_\sigma)^c$, and sends $(\tilde{g}, \tilde{h})$ to $\mathcal{A}$.
4. $\mathcal{S}$ receives $(u_0', w_0'), (u_1', w_1'), (u_2, w_2), (u_3, w_3), (u_4, w_4)$ from $\mathcal{A}$, like an honest R. Because in this case $j = 0$, $\mathcal{S}$ can decrypt $(u_3, w_3), (u_4, w_4)$ and gets (k_0^2, k_1^2), also $\mathcal{S}$ can decrypt $(u_0', w_0'), (u_1', w_1'), (u_2, w_2)$ and verifies the commitments. If there exists a commitment that does not hold, the $\mathcal{S}$ outputs $\perp$, otherwise, $\mathcal{S}$ can gets k_0^1, k_1^1, m.
5. $\mathcal{S}$ rewinds to the first step, and sets $h_1 = (g_1)^{a+1}$, which means that $j = 1$.
6. $\mathcal{S}$ interacts with $\mathcal{A}$ as and honest R, if $\mathcal{S}$ outputs $\perp$, then $\mathcal{S}$ rewinds to step 1 just like step4. Repeats this process until $\mathcal{S}$ extracts k_τ^1.
7. $\mathcal{S}$ compares k_τ^1 with k_0^1, k_1^1, then he can get τ.
8. $\mathcal{S}$ sends $k_0^1, k_1^1, k_0^2, k_1^2, m, \tau$ to the trusted party, outputs whatever $\mathcal{A}$ outputs and halts.

It is easy to see that the outputs of the ideal execution between $\mathbb{S}$ and an honest receiver R is identical to the outputs of a real execution with $\mathbb{A}$ and an honest honest receiver R.

Efficency: To construct the inputs of $CombinedRAND$(the inputs can be view as a public key to encryption scheme based on $RAND$), R computes 5 exponentiations; to construct the ciphertext, S computes 20 exponentiations and 10 multiplications; to decrypt the ciphertext, if $j = 0$, R computes 5 exponentiations, 5 modular inverses if $\sigma = 1$ or 6 modular inverses if $\sigma = 0$, 5 multiplications and 3 commitments verifications, if $j = 1$, R computes 3 exponentiations, 3 modular inverses, 3 multiplications and 2 commitments verifications, the expectation of computes are 4 exponentiations, 4.25 modular inverses, 4 multiplications and 2.5 commitments verifications. The protocol takes 3 rounds of communication, and the parties exchange 15 group elements (5 of them are elements of the public key, and 10 of them are ciphertext). It also need 1 zero knowledge proof of discrete logarithm.

6 Application in Secure Two-party Computation

6.1 Batch Cut-and-choose Bilateral OT

The protocol of the cut-and-choose bilateral OT in section 5.3 is designed only for one garbled circuit and the circuit has only two input wires, one is for P_1 and the other is for P_2. In this section, we extend it to batch cut-and-choose bilateral OT, that can be used in s garbled circuits and each circuit has $2n$ input wires, n of them are for P_1 and the other n of them are for P_2.

In order to simplify the description of the protocol, we define some vector symbols based on the notations in section 2.1. In the kth garbled circuit $G^k(C)$ $(k = 1, \ldots, s)$,

- Vector $\boldsymbol{k_0^{1,k}}$ is composed of all the keys corresponding to 0 in P_1's n input wires, that is $\boldsymbol{k_0^{1,k}} = \{k_0^{1,k}, \ldots, k_0^{i,k}, \ldots, k_0^{n,k}\}$, where $1 < i < n$;
- Vector $\boldsymbol{k_1^{1,k}}$ is composed of all the keys corresponding to 1 in P_1's n input wires, that is $\boldsymbol{k_1^{1,k}} = \{k_1^{1,k}, \ldots, k_1^{i,k}, \ldots, k_1^{n,k}\}$, where $1 < i < n$;
- Vector $\boldsymbol{m^k} = \{m^{1,k}, \ldots, m^{i,k}, \ldots, m^{n,k}\}$, where the elements $m^{i,k}$ indicates the order of the key $k_0^{i,k}$ and $k_1^{i,k}$;
- Vector $\boldsymbol{k_0^{2,k}}$ is composed of all the keys corresponding to 0 in P_2's n input wires, that is $\boldsymbol{k_0^{2,k}} = \{k_0^{n+1,k}, \ldots, k_0^{j,k}, \ldots, k_0^{2n,k}\}$, where $n + 1 < j < 2n$;
- Vector $\boldsymbol{k_1^{2,k}}$ is composed of all the keys corresponding to 1 in P_2's n input wires, that is $\boldsymbol{k_1^{2,k}} = \{k_1^{n+1,k}, \ldots, k_1^{j,k}, \ldots, k_1^{2n,k}\}$, where $n + 1 < j < 2n$;

The symbols can be described in Table 4.

Table 4. The random keys in s Garbled circuits

$G^1(C) \sim$	P_1's input wires: $w_1 \sim w_n$						P_2's input wires: $w_{n+1} \sim w_{2n}$					
$G^s(C)$	Symbol	w_1	$\cdots$	w_i	$\cdots$	w_n	Symbol	w_{n+1}	$\cdots$	w_j	$\cdots$	w_2n
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$G^k(C)$	$\boldsymbol{k_0^{1,k}} = \{$	$k_0^{1,k},$	$\cdots,$	$k_0^{i,k},$	$\cdots,$	$k_0^{n,k}\}$	$\boldsymbol{k_0^{2,k}} = \{$	$k_0^{n+1,k},$	$\cdots,$	$k_0^{j,k},$	$\cdots,$	$k_0^{2n,k}\}$
	$\boldsymbol{k_1^{1,k}} = \{$	$k_1^{1,k},$	$\cdots,$	$k_1^{i,k},$	$\cdots,$	$k_1^{n,k}\}$	$\boldsymbol{k_1^{2,k}} = \{$	$k_1^{n+1,k},$	$\cdots,$	$k_1^{j,k},$	$\cdots,$	$k_1^{2n,k}\}$
	$\boldsymbol{m^k} = \{$	$m^{1,k},$	$\cdots,$	$m^{i,k},$	$\cdots,$	$m^{n,k}\}$						
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Functionality: $\mathcal{F}_{\mathcal{BCACBOT}}$

- **Inputs:** The sender S's inputs are $x = \tau_1 \ldots \tau_n$, for $k = 1$ to s, $(\boldsymbol{k_0^{1,k}}, \boldsymbol{k_1^{1,k}}, \boldsymbol{m^k}, \boldsymbol{k_0^{2,k}}, \boldsymbol{k_1^{2,k}})$; the receiver R's input are $y = \sigma_1 \ldots, \sigma_n$ and $j = j_1 \ldots j_s$.
- **Auxiliary input:** Both parties hold a security parameter 1^n and $(\mathbb{G}, g_0, q)$, where $\mathbb{G}$ is a group of order q with a generator g_0, and q is of length n. The commitments $((com(\boldsymbol{k_0}^{1,1}), com(\boldsymbol{k_1^{1,1}}), com(\boldsymbol{m^1})), \ldots, (com(\boldsymbol{k_0^{1,s}}), com(\boldsymbol{k_1^{1,s}}), com(\boldsymbol{m^s}))$, where the commitments to a vector includes commitments to every components of the vector and the order of those commitments should be changed according to $\boldsymbol{m^k}$.
- **Output:**
  - The sender outputs nothing
  - The receiver outputs that

For $k=1$ to s

if $j_k=0$ **then** For $l=1$ to n outputs $(k_0^{l,k}, k_1^{l,k}, m^{l,k}), (k_0^{n+l,k}, k_1^{n+l,k})$;

else if $j_k=1$ **then** For $l=1$ to n outputs $(k_{\tau_l}^{l,k}, k_{\sigma_l}^{n+l,k})$;

Protocol 4: Batch Cut-and-choose Bilateral OT: $OT4$.

---

1. The receiver R chooses randomly $a \in \mathbb{Z}_q$, computes $h_0=(g_0)^a$ and send
2. For $k=1$ to s, R chooses randomly $b_k \in \mathbb{Z}_q$, computes $g_1^k=(g_0)^{b_k}$ and
   if $j_k=0$ **then** $h_1^k=(h_0)^{b_k}=((g_0)^a)^{b_k}$;
   else if $j_k=1$ **then** computes $h_1^k=(h_0)^{b_k+1}=((g_0)^a)^{b_k+1}$;
3. P sends $(h_0, g_1^1, h_1^1, \ldots, g_1^s, h_1^s)$ to S.
4. For $k=1$ to s, R proves to S that he know the discrete logarithm of g_1^k relative to g_0.
5. For $k=1$ to s, R chooses a random $c_k \in \mathbb{Z}_q$, and computes $\tilde{g}^k=(g_\sigma^k)^{c_k}, \tilde{h}^k=(h_\sigma^k)^{c_k}$, that is
   if $\sigma=0$ **then** computes $\tilde{g}^k=(g_0^k)^{c_k}, \tilde{h}^k=(h_0^k)^{c_k}$;
   else if $\sigma=1$ **then** computes $\tilde{g}^k=(g_1^k)^{c_k}, \tilde{h}^k=(h_1^k)^{c_k}$;
6. For $k=1$ to s
   For $l=1$ to n
   - S computes $((u_0^{l,k}, v_0^{l,k}), (u_1^{l,k}, v_1^{l,k}), (u_2^{l,k}, v_2^{l,k}), (u_3^{l,k}, v_3^{l,k}), (u_4^{l,k}, v_4^{l,k})) \leftarrow$ $CombinedRAND(g_0, h_0, g_1^k, h_1^k, \tilde{g}^k, \tilde{h}^k)$, and $w_0^{l,k}=v_0^{l,k}\cdot k_{\tau_l}^{l,k}$, $w_1^{l,k}=v_1^{l,k}\cdot k_{1-\tau_l}^{l,k}$, $w_2^{l,k}=v_2^{l,k}\cdot m^{l,k}$, $w_3^{l,k}=v_3^{l,k}\cdot k_0^{n+l,k}$, $w_4^{l,k}=v_4^{l,k}\cdot k_1^{n+l,k}$.
   - S permutes the order of $(u_0^{l,k}, w_0^{l,k}), (u_1^{l,k}, w_1^{l,k})$ randomly as $((u_0^{l,k})', (w_0^{l,k})'), ((u_1^{l,k})', (w_1^{l,k})')$,
   - S sends $((((u_0^{l,k})', (w_0^{l,k})'), ((u_1^{l,k})', (w_1^{l,k})'), (u_2^{l,k}, w_2^{l,k})(u_3^{l,k}, w_3^{l,k}), (u_4^{l,k}, w_4^{l,k}))$ to R.
7. For $k=1$ to s
   - If j_k=0, For $l=1$ to n
     - R computes $d_0^{l,k}=(w_0^{l,k})'/((u_0^{l,k})')^a$,$d_1^{l,k}=(w_1^{l,k})'/((u_1')^{l,k})^a$, $m^{l,k}=v_3^{l,k}/((u_3)^{l,k})^a$.
     - R verifies if $com(m^{l,k})$ is the commitment of $m^{l,k}$, and $com(k_{m^{l,k}}^{l,k})$ is the commitment of one of $d_0^{l,k}, d_1^{l,k}$, and $com(k_{1-m^{l,k}}^{l,k})$ is the commitment of the other one. If not, R outputs $\perp$, otherwise, he can rearrange the order of $com(k_{m^{l,k}}^{l,k}), com(k_{1-m^{l,k}}^{l,k})$ as $com(k_0^{l,k}), com(k_1^{l,k})$, and then rearrange the order of $d_0^{l,k}$ and $d_1^{l,k}$ as $k_0^{l,k}, k_1^{l,k}$.
     - If $\sigma_l=0$, computes $k_0^{n+l,k}=w_3^{l,k}/((u_3)^{l,k})^{c_k}$, $k_1^{n+l,k}=w_4^{l,k}/((u_4)^{l,k})^{c_k b_k^{-1}}$
     - ElseIf $\sigma_l=1$, computes $k_0^{n+l,k}=w_3^{l,k}/(u_3^{l,k})^{b_k c_k}$, $k_1^{n+l,k}=w_4^{l,k}/(u_4^{l,k})^{c_k}$
   - Elseif $j_k=1$ For $l=1$ to n
     - R computes $d_0^{l,k}=(w_0^{l,k})'/((u_0^{l,k})')^a$,$d_1^{l,k}=(w_1^{l,k})'/((u_1')^{l,k})^a$.
     - R verifies whether only one of $com(k_{m^{l,k}}^{l,k}), com(k_{1-m^{l,k}}^{l,k})$ is the commitment of one of $d_0^{l,k}$ and $d_1^{l,k}$, if not, R outputs $\perp$, otherwise it outputs the one in $d_0^{l,k}$ and $d_1^{l,k}$, which is correctly verified as $k_{\tau_l}^{l,k}$.
     - If $\sigma_l=0$, computes $k_0^{n+l,k}=w_3^{l,k}/(u_3^{l,k})^{c_k}$
     - ElseIf $\sigma_l=1$, computes $k_1^{n+l,k}=w_4^{l,k}/(u_4^{l,k})^{c_k}$

6.2 Secure Two-party Computation Based on Batch Cut-and-choose Bilateral OT

Based on the protocol of batch cut-and-choose bilateral OT, a secure two-party computation protocol can basically be divided into stages as follows:

- **Garbled circuit preparation:** P_1 constructs s copies of a Yao garbled circuit for computing the function.
- **Commitments preparation:** For each P_1's input wires, P_1 randomly selects permutation factor, permutes the key associated with it, and commits to the keys and permutation factor.
- **Oblivious transfer:** P_1 and P_2 perform the batch cut-and-choose bilateral OT protocol.
- **Circuit check:** P_2 verifies the correctness of the check-circuits.
- **Circuit evaluation:** P_2 computes the evaluation circuits and gets the output.

From the high level description of the protocol based on batch cut-and-choose bilateral OT, it is easily to see that the round complexity is decreased.

Acknowledgments. This work is supported by the National Natural Science Foundation of China under grant No.61173139, Key Project of Natural Science Foundation of Shandong province under grant No.ZR2011FZ005, and Doctoral Foundation of Ministry of Education of China under grant No.20110131110027. Thanks a lot to an anonymous reviewer of Asiacrypt 2014, who gives us some very useful advices.

References

1. C.Hazay and Y.Lindell. Efficient Secure Two-Party Protocols:Techniques and Constructions. Springer,November 2010.
2. O. Goldreich. Foundations of Cryptography: Volume 2 C Basic Applications. Cambridge University Press, 2004. University Press, 2004.
3. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game C A Completeness Theorem for Protocols with Honest Majority. In the 19th STOC, pages 218C229, 1987.
4. Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In EUROCRYPT 2007, Springer (LNCS 4515), pages 52-78, 2007.
5. Y. Lindell and B. Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In TCC 2011, Springer (LNCS 6597), pages 329-346, 2011.
6. Y. Lindell. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries. In CRYPTO 2013, Springer (LNCS 8043), pages 1-17.
7. Abhi shelat, Chih-hao Shen. Two-Output Secure Computation with Malicious Adversaries.Eurocrypt 2011, LNCS 6632, pp. 386C405, 2011.
8. B. Pinkas, T. Schneider,N. Smart, S. Williams. Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250C267.Springer, Heidelberg (2009)
9. Yan Huang, Johathan Katz and Davis Evans. Efficient secure Two-Party Computation using symmetric Cut-and-choose. In CRYPTO 2013, Springer (LNCS 8043), pages 18-35.
10. A Afshar, P Mohassel, B Pinkas, B Riva. Non-Interactive Secure Computation Based on Cut-and-Choose. Advances in CryptologyCEUROCRYPT 2014, 387-404.
11. Kiraz M S, Schoenmakers B, Villegas J. Efficient committed oblivious transfer of bit strings[M]//Information Security. Springer Berlin Heidelberg, 2007: 130-144.
12. C. Peikert, V. Vaikuntanathan and B. Waters. A Framework for Efficient and Composable Oblivious Transfer. In CRYPTO08, Springer (LNCS 5157), pages 554C571, 2008.
13. M. Kiraz and B. Schoenmakers. A Protocol Issue for the Malicious Case of Yaos Garbled Circuit Construction. In the Proceedings of the 27th Symposium on Information Theory in the Benelux, pages 283C290, 2006.
14. M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In 12th SODA, pages 448C457, 2001.
15. Y. Dodis, R. Gennaro, J. Hastad, H. Krawczyk and T. Rabin. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In CRYPTO 2004, Springer (LNCS 3152), pages 494-510, 2004.
16. A.C. Yao. How to Generate and Exchange Secrets. In the 27th FOCS, pages 162C167, 1986.